Development Plan for COMPSCI 4ZP6

RatBAT: Rat Behavioral Analysis Tool

Team 8

Brandon Carrasco

Daniel Locke

Jamie Wong

Inoday Yadav

Table of Contents

1 Revision History

Date	Version	Notes
10/25/2024	0	Original draft of document
04/04/2025	1	Final version of document

2 Team Meeting & Communication Plan

Communication between team members will typically be conducted using **Microsoft Teams**, a software that we are all familiar with using in an academic setting and which also has the added benefit of being the same platform we can use to communicate with the course instructor and TAs.

To keep in line with our planned workflow and maintain a good balance between fluidity and structure, our team will meet **at least twice a week** for approximately **15 to 30 minutes**. These meetings will occur **every Monday** at **11:00 AM** and **every Friday** at **4:00 PM**. The intention is for these meetings to allow us to set goals, impart progress, and raise issues at the beginning and end of the work week as check-in points to ensure that we all stay on track.

Although daily meetings would be preferable, after taking into account the sum total of our team's workload for other courses, it's been decided that keeping required meetings at these critical points would allow for the greatest productivity given our individual schedules. For any issues that require immediate attention, we have a **shared MS Teams group chat** that we may freely discuss the project on or schedule a meeting on.

Meetings with the **project's supervisors** will occur **at least biweekly**, allowing them the opportunity to examine fresh prototypes and provide feedback, raise issues or concerns about the direction of the project, etc. Meetings with the **course instructor and/or TAs** will be scheduled for **at least once per month**, with the intent to gather feedback on deliverables and advice on implementation and organizational details. Both of these meetings will ensure that our stakeholders needs are kept in mind and that our project does not veer off course.

For the project itself, we will be putting all goals, planning, issues, etc. on a **GitHub Project** board using a **Kanban** format. All deliverables, improvements, bugs, etc. will be added to the **Kanban board** with the appropriate labels, timelines, and assigned team members, allowing every team member to know what needs to be done in a visually nice way and as well as giving them the ability add to the board of their own volition.

Creation of project documentation will be done using **Google Docs**, an online collaborative text editor, and it shall be stored on **GitHub**, which has a version control system. In the future the option of **Confluence**, which possesses both features, will be explored should the need arise.

3 Team Member Roles

3.1 Assigned Roles & Responsibilities

Team Member	**Role**	**Main Implementation Responsibilities**
Brandon Carrasco	Project Manager, Developer	Summary Measures Package, Preprocessing Algorithms
Daniel Locke	Developer	Database, Preprocessing Algorithms
Jamie Wong	Developer	Web Platform
Inoday Yadav	Developer	Web Platform

3.2 Associated Requirements for Implementation

Legend	
Functional Requirements	**P0** = Priority 0 **P1** = Priority 1 **P2** = Priority 2 **P3** = Priority 3
Non-Functional Requirements	**LF** = Look and Feel Requirements **UH** = Usability and Humanity Requirements **P** = Performance Requirements **SP** = Security and Privacy Requirements **L** = Legal Requirements

Web Platform

- Associated Functional Requirements
  - **P0**.1, **P0**.2, **P0**.3, **P0**.5, **P0**.6, **P0**.7, **P0**.9, **P0**.10, **P0**.11
  - **P1**.1, **P1**.2, **P1**.4, **P1**.5
  - **P2**.1, **P2**.2, **P2**.3, **P2**.4
  - **P3**.1, **P3**.2
- Associated Non-Functional Requirements
  - **LF**.1, **LF**.2, **LF**.3
  - **UH**.1, **UH**.2, **UH**.3
  - **P**.1
  - **SP**.1, **SP**.2
  - **L**.1, **L**.2, **L**.3, **L**.4

Database

- Associated Functional Requirements
  - **P0**.4, **P0**.8, **P0**.12
  - **P1**.3, **P1**.4, **P1**.5
  - **P2**.2
- Associated Non-Functional Requirements
  - **P**.1
  - **SP**.1
  - **L**.4

Summary Measures Package

- Associated Functional Requirements
  - **P0**.7, **P0**.10, **P0**.11
  - **P2**.4
- Associated Non-Functional Requirements
  - **UH**.2
  - **P**.3
  - **L**.3

Preprocessing Algorithms

- Associated Functional Requirements
  - **P0**.10, **P0**.11
  - **P1**.1
- Associated Non-Functional Requirements
  - **UH**.2
  - **P**.2, **P**.4
  - **L**.3

4 Workflow Plan

Our overall workflow will be done in an **Agile-like** manner: **two-week sprints**, with each being started with a **sprint planning meeting** and ending with a **sprint review**. As we plan for an **iterative development process**, each sprint will produce a new **prototype** of the product. These prototypes will be **shown to our supervisors** to ensure that they understand our progress and that we receive actionable feedback from them to help adjust our product.

For the development process, we will be using **GitHub** (and the associated **GitHub Projects**) to store our project files and keep track of our goals, tasks, and deliverables. In terms of version control, the **main branch** will contain the **latest version of our product**, and at the end of a sprint, our latest prototype.

Every task will be analyzed for its dependencies on other tasks or components of the software. At the beginning of every new task (which usually corresponds to every sprint), a **new branch** will be created for the completion of that new task (usually an improvement of some kind,

assigned to a particular team member), unless that task is dependent on a **current task in-progress**, at which point the new task will be worked on in that **existing branch**. Generally, the latter case will be avoided as much as possible as it would depend on a volatile component and could delay the merge of that in-progress task, which, if new tasks depend on that component, could cause a feedback loop of delayed merges. Once a task has been completed, it shall be **unit tested**, have its **bugs ironed out**, **merged** into the main branch, **unit tested** again, and any **bugs found will be ironed out**.

Should data be required to be stored during development and testing, it shall be stored within a **database** hosted on a McMaster **server** or a team member's personal machine before the **server** is set up. This **server** and team member's personal machine will also be used for resource-intensive computations. **Every sprint**, **upon completion** of development/testing of a task, any data used for its development/testing shall be **deleted** from the **server**, unless the data is **tied to a bug** that needs to be fixed.

For measurement of performance metrics relating to our web platform including **Time to First Byte** response time metric we will use Google's **Chrome DevTools**. In order to test the performance of our **SPSM** smoothing algorithm we will implement our proposed benchmark smoothing algorithms (**MA**, **LP**, and **LOWESS**) and use sample **time series data** to compare performance. Accuracy of partitioning will be computed using the **Python Sci-Kit Learn** library.

Tasks, bugs, and other pieces of work will be created on the **GitHub Project** associated with our project. Priority, type of task (mother task, subtask, or atomic task), manpower hours required, planned start and end, and team member assigned will be added to the **GitHub Issue**, as well as a brief description of what the task involves, including a checklist of items to complete. These Issues will move through the typical **Kanban-style board**, going from **backlog**, **design**, **to-do**, **in-progress**, **review**, and then finally **completion**. Team members are responsible for updating the **GitHub Issues** they're assigned to throughout development.

<u>To summarize</u>:

1. Sprint planning meeting is held at the beginning of a two-week sprint, where new tasks, bugs, etc. will be discussed.
    a. These will be turned into GitHub Issues, labeled appropriately, and assigned to a team member for completion.
2. New branches will be created wherever necessary, and team members will begin work.
3. Check-in meetings will provide avenues for discussion of impediments, progress, etc.
4. Upon task completion:
    a. Unit test branch and fix any bugs found.
    b. Notify the team, allow for a quick code review, then merge into the main branch.
    c. Unit test main branch and fix any bugs found.

d. If any used data is no longer valid (used for development/testing/bug fixing), then delete it from the database.

5. At the end of the Sprint, review the main branch's code, bugs, etc., and discuss progress, issues, and how the process can be improved for the next sprint.

6. Display prototype (main branch product) to supervisors to gather their feedback and revise plan appropriately if need be.

5 Proof of Concept Demonstration Plan

Our proof of concept will consist of the general framework and functionality of our web app which will demonstrate how our product will work and how all our requirements will come together to form our product.

We will build a barebones version of our user interface which will demonstrate the base features available to the user. This will include a filtering pane that will allow users to slice the data, the ability to view tabular time series data and an interface to apply smoothing and segmentation as well as base summary measures to the data. We will also have several example data visualization components which will help illustrate the role data visualization will play in our final product.

To facilitate this interface we will build the basics of our backend. The PoC version of our backend will be able to fetch data from the FRDR via Globus, and store that data in a temporary database set up on a local machine. We also plan to have a few simple summary measures implemented to serve as a demonstration of this component of our project. For some of the features that will not be implemented for the purposes of the PoC, such as the smoothing and segmentation component, we will have example files that simulate the input and output of these phases to facilitate the demonstration of the user interface.

6 Technology

6.1 Programming Languages & Libraries

Coding environment:

- Visual Studio Code

Frontend:

- **Languages:**
  - JavaScript, HTML, CSS
- **UI libraries/Framework:**
  - React

Backend:

- **Languages:**
  - Python, SQL
- **Framework:**

    - Django
- **DBMS:**
    - MariaDB
- **Libraries:**
    - Pandas, Numpy, Scikit-Learn

6.2 Performance & Testing Tools

We will use **PyTest** along with **pytest-django** as the primary testing framework for the **backend**, in addition to **Jest** for the **frontend**. PyTest allows us to create **isolated tests** for **each function**, ensuring that individual components behave as expected, especially when handling large datasets or sensitive behavioral data. Pytest-django allows **integration tests** to run smoothly, particularly for database interactions, API endpoints, and data workflows. Jest will be used to test **React components**. We'll write **unit tests** for **each component** to ensure they render and function correctly, such as handling input, displaying data, and allowing interactions like data filtering and visualization.

7 Coding Standard

- All Python code will be written in adherence to the standard set out in the PEP8 style guide.
- All JavaScript and React code will be written in adherence to the Airbnb style guide.
- All sensitive data shall be stored securely through AES encryption standards.

8 Project Scheduling

Below is a Gantt chart depicting the general timeline for the development of our project, from planning to testing & final polishing.

	Sep. 2024	Oct. 2024	Nov. 2024	Dec. 2024	Jan. 2025	Feb. 2025	Mar. 2025
Planning							
Research							
Implementation							
-> Web Platform							
-> Database							
-> Summary Measures Package							
-> Preprocessing Algorithms							
Testing & Polishing							